Decoupling Provenance Capture and Analysis from Execution

Stamatogiannakis, M.; Groth, P.T.; Bos, H.J.

published in
7th USENIX Workshop on the Theory and Practice of Provenance (TaPP'15)
2015

document version
Publisher's PDF, also known as Version of record

Decoupling Provenance Capture and Analysis from Execution

Manolis Stamatogiannakis *
VU University Amsterdam
manolis.stamatogiannakis@vu.nl

Paul Groth
Elsevier Labs
p.groth@elsevier.com

Herbert Bos
VU University Amsterdam
h.j.bos@vu.nl

Abstract

Capturing provenance usually involves the direct observation and instrumentation of the execution of a program or workflow. However, this approach restricts provenance analysis to *pre-determined* programs and methods. This may not pose a problem when one is interested in the provenance of a well-defined workflow, but may limit the analysis of unstructured processes such as interactive desktop computing. In this paper, we present a new approach to capturing provenance based on full execution record and replay. Our approach leverages full-system execution trace logging and replay, which allows the complete decoupling of analysis from the original execution. This enables the *selective* analysis of the execution using *progressively* heavier instrumentation.

Keywords provenance, introspection, reverse engineering

1. Introduction

A key challenge when developing provenance aware systems is deciding what provenance should be captured during the execution of that system. Developers need to ensure that enough provenance is captured such that future analysis will be able to be performed. If a developer fails to foresee the need for a particular kind of provenance to be captured (e.g. not capturing the loading of libraries), then subsequent analysis will be difficult and would require one to attempt to reconstruct the missing provenance [10, 15, 22].

One answer, then, is to capture all relevant provenance information. However, we may not be able to determine this a priori. In some cases, an entire system is deterministically replayable and all operators of importance are known in advance. This is an approach adopted within database systems [6, 13] where the operators in question are well-known and have clear semantics. Such a replay centric approach has also be recently applied to big data analysis systems [20]. However, many systems, for example desktop computing environments, have operations which are complex or unknown beforehand and thus thus have unclear semantics.

For other applications, one could apply software methodology to help make informed decisions upfront about what provenance information should be captured and the level of instrumentation required for that. Miles et al. created such software methodology called PrIME [21]. Unfortunately, even such a systematic approach cannot foresee every possible analysis.

Finally, lightweight always-on provenance capture can be employed. The runtime overhead of provenance capture can vary between 1-100% depending on the system used [2]. For operating system based capture, state of the art systems have roughly a 10% overhead [11]. While the runtime overhead can be kept relatively low, several compromises have to be made to achieve this: *a*) manual instrumentation effort may be required for each application; *b*) the produced provenance may not semantically rich [30]; *c*) virtually, no analysis flexibility is offered; and *d*) intensive provenance analysis [24] is precluded.

In this work, we *eliminate the need for developers to decide a priori what provenance needs to be captured*. We do this by leveraging full system execution trace logging and replay, which can be done at a fixed, relatively small overhead. In essence, our approach *replays* exactly what a system performs adding provenance instrumentation needed for a particular analysis after the fact. More importantly, our approach enables *iterative/user-driven* provenance analysis during replay, using progressively heavier instrumentation. The user or analyst can use provenance queries to find portions of the execution trace to focus on and to request such additional instrumentation.

The contributions of the paper are as follows:

- A 4-stage methodology for selectively and progressively applying provenance capture instrumentation in the context of an execution record/replay environment.
- Two plugins for the PANDA record/replay environment which can be used for provenance analysis using our methodology.
- A demonstration of the application of the system using a case study.

The rest of the paper is organized as follows. We begin by describing our methodology and system. This is followed by a case study description. We then discuss the ramifications for overall system performance and alternative methods of record and replay. Finally, we conclude.

2. System and Methodology

Our system is based on the following, 4-stage methodology:

1. **Execution Capture**: At runtime, a self-contained replayable execution trace is captured.
2. **Application of instrumentation**: In this stage, depending on goal of the analysis, an instrumentation plugin is selected to process the execution trace. Through this process, a provenance graph is generated.
3. **Provenance analysis**: In this stage, the user interrogates the provenance graph using a query language to focus on and/or select portions of the graph.
4. **Selection and iteration**: Based on the provenance analysis, the user can select a portion of the execution trace to apply

* Part of this work was done while the author was visiting SRI International's Computer Science Laboratory.

This article was originally published in *Proceedings of the 7th USENIX Workshop on the Theory and Practice of Provenance* (TaPP 2015).

additional, more intensive, instrumentation on. To do this, the user starts again from stage 2.

Before describing our system, we provide a brief background on execution record and replay that is at the heart of our system.

2.1 Full System Record and Replay

Recording and replaying of full system executions became popular in the early years of this millennium with applications almost exclusively related to debugging and security/intrusion analysis [9, 26]. After all, the ability to replay an execution with limited overhead is ideally suited to finding rare and non-deterministic error conditions and for analysing attacks as they occur. Moreover, by decoupling the analysis from the execution the overhead can be kept to a minimum [4, 23]. The functionality became so popular that VMware started shipping it in some of its products [27]. Over time the techniques became more efficient [3], but the application domains remained the same. Here, we show the usefulness of these techniques in the field of provenance.

2.2 The PANDA analysis framework

The *Platform for Architecture-Neutral Dynamic Analysis* (PANDA) [8] is an open source framework for full-system analysis based on execution record and replay. It is based on the QEMU emulator [1]. PANDA recording works by taking a snapshot of the guest memory and subsequently recording all the non-deterministic inputs to the guest CPU. The recorded information enables the *deterministic replay* of all the execution at any later time.

Because recoding happens at the "boundaries" of the virtual CPU, execution traces recorded with PANDA cannot be used to "go live" as they do not include the state of the emulated hardware devices. This is a conscious decision in the design of PANDA, as it is meant as an analysis framework rather than a generic VM. Another property of PANDA execution traces is that they are *self-contained* (i.e. you can replay and analyze a trace without having access to the VM image that was used to record it).

One key feature of PANDA is its plugin architecture which allows writing analysis modules in C and C++. The plugins can insert instrumentation at different granularities: per instruction, per memory access, per context switch etc. More importantly, PANDA offers a framework for the plugins to interact with each other. This allows implementing complex analysis functionality by composing it from several smaller plugins. This approach follows the Unix philosophy of relying on combining a plethora of small tools, rather than building monolithic end-all, be-all tools. PANDA allows plugins to either invoke functionality of other plugins through *API calls*, or register plugin-specific *callbacks*.

2.3 The PROV-Tracer plugin

PROV-Tracer is the central PANDA plugin we developed for supporting provenance analysis. The plugin taps on `osi` and `osi_linux` plugins for some functionality. Most of this functionality was developed from scratch to support PROV-Tracer but it was modeled as separate plugins because it was not provenance-specific. In fact we have reports of researchers using it for other types of analysis.

The operation of PROV-Tracer can be summarized as:

1. Registering for notifications on the creation/destruction of processes. The callback mechanism is implemented by the `osi` plugin.
2. Retrieving process information from guest OS memory for each new process. The guest OS introspection is implemented by the `osi_linux` plugin.
3. Decoding and keeping track of the performed system calls. This functionality is embedded in `prov_tracer` plugin.
4. Doing book-keeping on the file usage, reads and writes performed by processes and emitting the respective provenance along the way.

In principle, PROV-Tracer works similar to existing provenance systems that monitor the OS-process interface for provenance-related events [10, 12, 16]. However, in contrast to those systems, PROV-Tracer does not have primary access to these events. Instead, it has to construct them from lower level semantics. E.g. a write to the `CR3` register signifies a context switch, which may (or may not) coincide with the creation of a new process.

PROV-Tracer emits provenance in a compact intermediate format. This choice was made to reduce the complexity of the required book-keeping. E.g. deduplication of the produced provenance relations is much easier to perform after the analysis has completed. A python script (`raw2ttl.py`) is used to convert the provenance to PROV serialized as Turtle RDF.

2.4 The ProcStrMatch plugin

The ProcStrMatch plugin monitors all memory accesses in the guest VM, waiting for a specific text string to appear. Its functionality is clearly simpler than that of the `prov_tracer` plugin. Although this type of analysis would not be adequate on its own for fully-fledged provenance analysis, it can prove very useful in enriching already extracted provenance. This is a demonstration of how our methodology enables the use of more intensive techniques not traditionally associated with provenance analysis.

The plugin is implemented on top of the `stringsearch` plugin, which offers the string matching functionality and the `osi`, `osi_linux` plugins which provide information about the currently running process at the time a match occurs. Because of the use of the `stringsearch` plugin, ProcStrMatch is particularly heavyweight in its analysis. The reason is that while `stringsearch` has carefully been designed to match strings as efficiently as possible (see [7]), it is still very heavyweight to use. This is because *a*) it has to be invoked very frequently – on every memory access, and *b*) some QEMU runtime optimizations have to be turned off to enable this. The plugin emits any matches as `:hasMemText` triples which can be merged with the results of PROV-Tracer and subsequently be queried in a unified way (see Section 3.4.1).

2.5 User-driven Provenance Analysis

As discussed previously, provenance produced by the system is represented using the W3C PROV recommendations[14] and serialized using Turtle RDF. This enables us to leverage existing Semantic Web infrastructure for provenance analysis. The integration with this generic infrastructure is done by specifying query result formats for obtaining parameter values that can then be fed back into the replay and instrumentation system (i.e. PANDA). For instance, to define a particular time range for future analysis, the SPARQL query returns a start and end time. An example query would be as follows:

```
SELECT ?stime ?etime WHERE {
  <file:/home/panda/letter.txt>
           prov:wasGeneratedBy ?act.
```

The captured inputs include CPU port I/O, DMA, hardware interrupts.

Source code available on:
https://github.com/m000/panda/tree/prov_tracer/

The prefixes used in the SPARQL queries reported here are defined by the RDFa Core Initial Context - http://www.w3.org/2011/rdfa-context/rdfa-1.1. We also define or own predicates in the `dt:` namespace.

```
  ?act prov:startedAtTime ?stime .
  ?act prov:endedAtTime ?etime .
}
```

This finds the activity that generated the letter.txt file and retrieves the start and end times of the activity. The use of SPARQL in combination with PROV provides a full featured language for users to interrogate and analyze the provenance produced by PROV-Tracer. Note, in the case study that follows we used the Stardog triple store, which supports SPARQL 1.1.

3. Case Study

To demonstrate the applicability of our approach, we apply our approach to the following scenario.

3.1 Scenario

Alice has been tasked by *Bob* to write a report on Camelidae. The sources of the report are sent to her as a tar archive named `camelidae.tar`.

1. The archive is extracted and four text files are generated: `alpaca.txt`, `bactrian.txt`, `guanco.txt`, and `llama.txt`.
2. Alice creates a new file with `vim` text editor. Some text is typed and the first three of the files are read into the buffer. The buffer is saved as `camelidae-report.txt`
3. Alice now edits `llama.txt` and adds some text to it before appending it to report. But Alice is now getting hungry, so she closes her editor and goes out to lunch.
4. Taking advantage of the author's absence, *Mallory* edits `llama.txt` adding the text: "Llamas are lame!".
5. Alice resumes editing and appends file `llama.txt` to it, without noticing the inserted text.
6. The report and its sources are packed into a new archive (`camelidae-new.tar`) and sent to Bob for proof-reading.

During proof-reading, Bob detects the inserted text and requests an analysis to identify the source. Fortunatelly, the whole editing process had been recorded in the execution trace `alice.et`.

3.2 Initial Provenance Analysis

To start the analysis, a quick full-system provenance analysis was produced by PROV-Tracer:

```
$ qemu -replay alice.et -panda
  "osi;osi_linux;prov_tracer"
$ ./raw2ttl.py < prov_out.raw > alice.ttl
```

We can see that PROV-Tracer stacks on top of two other PANDA plugins. The `osi` plugins enable it to introspect the running OS and get information on the running processes. On top of that, PROV-Tracer decodes all the system calls that were executed and associates them with a process, as described in Section 2.3.

Because all the running processes are tracked, the quick analysis of PROV-Tracer tends to produce unwieldy provenance graphs, even for a short scenario like the one we study. The provenance graph produced by the plugin for this scenario has 3063 triples and 222 unique `prov:Activities`.

As a first step, a query was written to see the text files from which `camellidae-new.tar` was derived:

```
SELECT ?file WHERE {
  <file:/home/panda/work/camelidae-new.tar>
  prov:wasDerivedFrom ?file .
  FILTER regex(str(?file), "txt")
}
```

```
Results:
file
file:/home/panda/work/alpaca.txt
file:/home/panda/work/bactrian.txt
file:/home/panda/work/guanco.txt
file:/home/panda/work/llama.txt
file:/home/panda/work/camelidae-report.txt
```

This verified that the `camelidae-report.txt` file edited on Alice's machine was indeed included in the archive submitted to Bob.

3.3 Selective Analysis

We now have an entry point to the part of the provenance graph of interest. Using this, the full execution trace can be trimmed down to only cover the time when processes interacting with the above files were active. For trimming down the trace, we use the pseudo-timestamps produced by PROV-Tracer (see Section 2.3). The pseudo-timestamps are extracted from the provenance trace using a SPARQL query. The following query extracts the relevant start and end positions of the editing sessions that touched the files in question.

```
SELECT (MIN(?startTime) AS ?s)
       (MAX(?endTime) AS ?e) WHERE
{
  <file:/home/panda/work/camelidae-new.tar>
  prov:wasDerivedFrom ?file .
  FILTER regex(str(?file), "txt")

  ?file prov:wasGeneratedBy ?activity .
  ?activity a dt:Editor .
  ?activity prov:startedAtTime ?startTime .
  ?activity prov:endedAtTime ?endTime .
}
```

```
Results:
s         e
705412095 990055363
```

Note, the above query shows the advantages of using an existing query language. We are able to make use of the existing aggregation and filtering functionality built into SPARQL. The results of the query can then be used to retrieve a portion of the trace using the `scissors` PANDA plugin.

```
$ export s=705412095 e=990055363
$ qemu -replay alice.et -panda
  "scissors:start=$s,end=$e,name=report.et"
```

The resulting execution trace `report.et` contains the smaller slice.

3.4 Progressive Analysis

Now that we have a more compact execution trace, it becomes possible to start applying more heavyweight instrumentation methods, which enable more detailed provenance analysis. The intensiveness of these methods stems from the fact that they apply instrumentation on a finer granularity (e.g. per memory access) than the initial quick analysis which operates on a per context switch granularity.

---

http://stardog.com

As an aside, Stardog is a versioned database that supports querying over its revisions using W3C PROV.

We note that we have broken the semantics of `prov:startedAtTime` and `prov:endedAtTime` as this should be an XML Schema date time. In future revisions, we will make changes to the correct formatting.

3.4.1 String Matching

One particularly heavy analysis is offered by the `stringsearch` PANDA plugin. The plugin instruments *all* the memory accesses in the trace (reads and writes) and attempts to match them against a list of strings, read from file. However, by itself the plugin does not provide high level process semantics. For this, we do not use it directly, but rather through the `psstrmatch` plugin which combines information from different sources to tell us the name of the processes that read/wrote a specific string to/from memory.

```
$ echo '"Llamas are lame"' > search_strings.txt
$ qemu -replay report.et -panda
  "osi;osi_linux;callstack_instr;stringsearch;psstrmatch"
```

By running this, we can determine the processes that wrote the infringing string to memory. The plugin outputs this information in file `smatch.ttl` which is in Turtle RDF format. As such, the new information can be loaded in the same triple store with the original trace to augment the existing graph. This is another benefit of using RDF, which allows for the incremental addition of data. After this update, we can query the provenance to retrieve the first of these processes using the following SPARQL query:

```
SELECT (MIN(?st) as ?min) ?vi WHERE
{
    ?vi dt:hasMemText "Llamas are lame".
    ?vi prov:startedAtTime ?st
} GROUP BY ?vi
```

```
Results:
min       vi
705412095 exe://vi~3547
782648505 exe://vi~3557
857809758 exe://vi~3570
963886071 exe://vi~3595
```

The first of these processes is the one which introduced the infringing text into the report editing pipeline. The process can be easily mapped to Mallory's username. This result is not presented in the query output because it is not currently supported by the PANDA `osi_linux` plugin. It is however straightforward to extend the `osi_linux` plugin so that this information is made available to PROV-Tracer.

With this case study, we have demonstrated the application of our methodology and the potential benefits of a progressive and decoupled analysis.

4. Discussion

4.1 Additional Analysis

With our analysis described, we managed to track the source of the infringing text back to a `vi` process belonging to Mallory. However can we be sure that Mallory hasn't introduced any other text in the report? This kind of question could be answered by applying *taint analysis* on the captured execution trace, as proposed in [24].

Taint analysis in general is prohibitively expensive for runtime application. Chow et al. [4] quote a 100x slowdown on their system, also based on QEMU. More recently, for security applications, Kemerlis et al. [18] have managed to reduce the overhead of taint analysis to the 4x-6x range for analysis of individual CPU-bound processes. For taint analysis tailored for provenance, as implemented in [24], the overhead is in the O(100x) range.

By applying taint analysis, very fine grained provenance can be produced, attributing the source of each byte in the final report back to a specific input source. This would help verify that Mallory didn't introduce any additional text that went unnoticed. It would also help to selectively "clean-up" the original provenance graph from potential false-positives. We are currently investigating the integration of the existing taint-analysis module of PANDA into our provenance analysis.

Another potential fine grained analysis is the application of program slicing to infer hidden provenance relationships [29]. As with taint tracking this approach allows for the elimination of false positives but is too expensive to be applied at runtime, with a reported 7x-40x slowdown.

In addition, to these analyses, one can imagine the addition of post hoc instrumentation to output additional semantics as we have have done by outputting that `vi` activities are a type of editing activity.

4.2 Performance

In a record/replay environment, the primary performance concern is low overhead during recording. Speed of replaying and analysis are of secondary importance, as they take place offline. The performance of PANDA proved adequate for paced user interaction with the VM. Yet, it still is insufficient for deployment in a production day-to-day environment The reason for the subpar performance is mainly the QEMU platform which incurs a 5x slowdown compared to native execution. On top of that, PANDA recording imposes an additional 10-20% overhead [7].

However, after paying this "startup cost", arbitrary types of provenance analysis may be explored. E.g. taint analysis for capturing provenance can easily incur an O(10x) slowdown. However, it has been argued [8] that performance of PANDA can be significantly improved by enabling KVM acceleration [19] on QEMU. However, this means that recording would have to be re-implemented inside the KVM module – a substantial undertaking.

4.3 Space Requirements

Another concern for whole-system execution recording is the space required to store the captured execution trace. The raw trace produced by PANDA for our case study scenario was roughly 160MB. This breaks down to a 111MB initial memory dump and a 49MB log of the non-deterministic inputs. We would expect the size of the non-deterministic input log would be much bigger in a loaded multi-tasking system. Although a more detailed study is needed to explore the exact space requirements for different types of systems, we believe that these requirements would still be within reach of todays' technology. Furthermore, several approaches could be taken for managing or reducing the space requirements for storing execution traces.

Compression: PANDA already includes a packing script for reducing the size of the captured execution traces and making sharing them easier. The script utilizes XZ compression [5] and provides substantial gains in terms of the space required to store the trace. For our case study, the total size of the trace was reduced to 32MB.

Hard disk rotation: The space required to capture the whole execution trace for a *working-day session* seems to fall well within the limits of today's desktop drive technologies, where 2TB-3TB drives are commonplace. This means that by adopting a hard disk drive rotation scheme (similar to rotating backup tapes), it would be possible to store several days worth of execution history at an affordable cost.

VM snapshots: An important design decision in the development of PANDA was that the produced traces must be *self-contained* in order to enable sharing among the research community. This decision has as consequence that the non-deterministic input log must also include any data read from the VM disk. If we don't need to share the produced traces, we could make them more compact by associating them with a VM disk snapshot. This would allow us to exclude any data read from disk from the non-deterministic input

log. On the downside, this would add significant complexity to PANDA, as now it would also have to deal with the disk emulation code of QEMU.

4.4 Alternative VM Introspection Methods

In order to introspect the guest VM, our prototype system relies on knowing the memory layout and data structures of the Linux kernel. Using this knowledge, the current state of the running OS can be reconstructed. However, because the kernel contains several conditionally compiled code fragments, the *exact* layout of its data structures (i.e. the exact offset of each member of the data structure) depends on the compile-time configuration of the kernel. These offset values can be easily acquired for the kernels of all popular distributions and loaded as a *kernel offsets profile* to our system. A similar profile-based approach is used by the Volatility memory forensics framework [25].

A different approach to introspection is employed by the TEMU [28] analysis framework, which uses a *guest driver* to export information from the guest OS. However, this approach is not applicable in our system, because PANDA cannot "go live" so that we can query the driver for the desired information.

Finally, Jones et al. [17] present a low-overhead method for detecting process creation in the guest OS. Their method however, is limited to the retrieval of the address space identifiers (*ASID*s) of the processes. As such, it is of limited use for our system as we seek to extract much richer information.

5. Conclusion

In this paper, we have shown how the application of record and replay techniques allows provenance analysis to be decoupled from execution. We described a four stage methodology for the interweaving of provenance analysis and progressively more detailed instrumentation. This methodology is realized in a system implemented using the combination of provenance specific PANDA plugins and Semantic Web infrastructure. The resulting system is demonstrated against a case study run in a desktop computing environment.

There are several avenues for future work. The application of provenance analysis to real world execution traces, for example those available at PANDA Share, to discover regularities. The improvement of the system through the development of additional plugins, more seamless integration of querying and instrumentation as well as performance improvements. On the usability side, it would be interesting to explore building a GUI shell over the current prototype. This would both make the system more accessible to analysts with little experience with SPARQL and would allow the visualization of provenance and debugging any anomalies. Finally, we aim to investigate the role that offline or cloud processing of provenance can play based on execution traces.

Overall, we believe that record/replay provides a powerful infrastructure for provenance capture and analysis that frees developers from upfront instrumentation concerns.

References

[1] F. Bellard. QEMU, a fast and portable dynamic translator. In *Proceedings of USENIX ATC'05*, Anaheim, CA, USA, Apr. 2005. URL http://dl.acm.org/citation.cfm?id=1247401.

[2] L. Carata, S. Akoush, N. Balakrishnan, T. Bytheway, R. Sohan, M. Selter, and A. Hopper. A primer on provenance. *Communications of the ACM*, 57(5):52–60, 2014. ISSN 0001-0782. . URL http://dx.doi.org/10.1145/2596628.

[3] Y. Chen and H. Chen. Scalable deterministic replay in a parallel full-system emulator. In *Proceedings of ACM SIGPLAN PPoPP'13*, Shenzhen, China, Feb. 2013. . URL http://dx.doi.org/10.1145/2442516.2442537.

[4] J. Chow, T. Garfinkel, and P. M. Chen. Decoupling dynamic program analysis from execution in virtual environments. In *Proceedings of USENIX ATC'08*, Boston, MA, USA, Jun. 2008. URL http://dl.acm.org/citation.cfm?id=1404015.

[5] L. Collin. XZ Utils: Free general-purpose data compression software with high compression ratio. http://tukaani.org/xz/. [Online; accessed Jun. 2015].

[6] Y. Cui and J. Widom. Lineage tracing for general data warehouse transformations. *The VLDB Journal*, 12(1):41–58, May 2003. ISSN 1066-8888. . URL http://dx.doi.org/10.1007/s00778-002-0083-8.

[7] B. Dolan-Gavitt, T. Leek, J. Hodosh, and W. Lee. Tappan Zee (North) Bridge: Mining memory accesses for introspection. In *Proceedings of the ACM SIGSAC CCS'13*, Berlin, Germany, Nov. 2013. . URL http://dx.doi.org/10.1145/2508859.2516697.

[8] B. Dolan-Gavitt, J. Hodosh, P. Hulin, T. Leek, and R. Whelan. Repeatable reverse engineering for the greater good with PANDA. Technical Report CUCS-023-14, Columbia University, Oct. 2014. URL http://ezid.cdlib.org/id/doi:10.7916/D8WM1C1P.

[9] G. W. Dunlap, S. T. King, S. Cinar, M. A. Basrai, and P. M. Chen. ReVirt: Enabling intrusion analysis through virtual-machine logging and replay. In *Proceedings of USENIX OSDI'02*, Boston, MA, USA, Dec. 2002. . URL http://dx.doi.org/10.1145/1060289.1060309.

[10] J. Frew, D. Metzger, and P. Slaughter. Automatic capture and reconstruction of computational provenance. *Concurr. Comput.: Pract. & Exper.*, 20(5):485–596, Apr. 2008. ISSN 1532-0634. . URL http://dx.doi.org/10.1002/cpe.1247.

[11] A. Gehani and D. Tariq. SPADE: Support for Provenance Auditing in Distributed Environments. In *Proceedings of Middleware 2012*, Montreal, Quebec, Canada, Dec. 2012. . URL http://dx.doi.org/10.1007/978-3-642-35170-9_6.

[12] E. Gessiou, V. Pappas, E. Athanasopoulos, A. Keromytis, and S. Ioannidis. Towards a universal data provenance framework using dynamic instrumentation. volume 376 of *IFIP Advances in Information and Communication Technology*, pages 103–114. 2012. ISBN 978-3-642-30435-4. . URL http://dx.doi.org/10.1007/978-3-642-30436-1_9.

[13] B. Glavic, K. S. Esmaili, P. M. Fischer, and N. Tatbul. Efficient stream provenance via operator instrumentation. *ACM Transactions on Internet Technology*, 14(1):7:1–7:26, Aug. 2014. ISSN 1533-5399. . URL http://dx.doi.org/10.1145/2633689.

[14] P. Groth and L. Moreau (eds.). PROV-Overview: An overview of the PROV family of documents. W3C Working Group Note NOTE-prov-overview-20130430, World Wide Web Consortium, Apr. 2013. URL http://www.w3.org/TR/2013/NOTE-prov-overview-20130430/.

[15] P. Groth, Y. Gil, and S. Magliacane. Automatic metadata annotation through reconstructing provenance. In *Proceedings of the 3rd International Workshop on the role of Semantic Web in Provenance Management*, Heraklion, Greece, May 2012.

[16] D. A. Holland, M. I. Seltzer, U. Braun, and K.-K. Muniswamy-Reddy. PASSing the provenance challenge. *Concurr. Comput.: Pract. & Exper.*, 20(5):531–540, Apr. 2008. ISSN 1532-0634. . URL http://dx.doi.org/10.1002/cpe.1227.

[17] S. T. Jones, A. C. Arpaci-Dusseau, and R. H. Arpaci-Dusseau. Antfarm: Tracking processes in a virtual machine environment. In *Proceedings of USENIX ATC'06*, Boston, MA, USA, May 2006. URL http://dl.acm.org/citation.cfm?id=1267360.

[18] V. P. Kemerlis, G. Portokalidis, K. Jee, and A. D. Keromytis. libdft: Practical Dynamic Data Flow Tracking for Commodity Systems. In *Proceedings of VEE'12*, London, UK, Mar. 2012. . URL http://dx.doi.org/10.1145/2151024.2151042.

[19] A. Kivity, Y. Kamay, D. Laor, U. Lublin, and A. Liguori. kvm: the Linux virtual machine monitor. In *Proceedings of the Linux Symposium*, Ottawa, Ontario, Canada, Jun. 2007.

http://www.rrshare.org

[20] D. Logothetis, S. De, and K. Yocum. Scalable lineage capture for debugging DISC analytics. In *Proceedings of ACM SOCC'13*, Santa Clara, CA, USA, Oct. 2013. . URL http://dx.doi.org/10.1145/2523616.2523619.

[21] S. Miles, P. Groth, S. Munroe, and L. Moreau. PrIMe: A Methodology for Developing Provenance-Aware Applications. *ACM TOSEM*, 20 (3):8:1–8:42, Aug. 2009. ISSN 1049-331X. . URL http://dx.doi.org/10.1145/2000791.2000792.

[22] T. D. Nies, S. Coppens, D. V. Deursen, E. Mannens, and R. V. D. Walle. Automatic discovery of high-level provenance using semantic similarity. In *Proceedings of IPAW'12*, Santa Barbara, CA, USA, Jun. 2012. . URL http://dx.doi.org/10.1007/978-3-642-34222-6_8.

[23] G. Portokalidis, P. Homburg, K. Anagnostakis, and H. Bos. Paranoid Android: Versatile protection for smartphones. In *Proceedings of ACSAC'10*, Austin, TX, USA, Dec. 2010. . URL http://dx.doi.org/10.1145/1920261.1920313.

[24] M. Stamatogiannakis, P. Groth, and H. Bos. Looking inside the black-box: Capturing data provenance using dynamic instrumentation. In *Proceedings of IPAW'14*, Cologne, Germany, Jun. 2014. . URL http://dx.doi.org/10.1007/978-3-319-16462-5_12.

[25] A. Walters. Volatility: An Advanced Memory Forensics Framework. http://www.volatilityfoundation.org/. [Online; accessed Jun. 2015].

[26] M. Xu, R. Bodik, and M. D. Hill. A "flight data recorder" for enabling full-system multiprocessor deterministic replay. In *Proceedings of ACM ISCA'03*, San Diego, CA, USA, Jun. 2003. . URL http://dx.doi.org/10.1145/859618.859633.

[27] M. Xu, V. Malyugin, J. Sheldon, G. Venkitachalam, and B. Weissman. ReTrace: Collecting execution trace with virtual machine deterministic replay. In *Proceedings of MoBS'07*, San Diego, CA, USA, Jun. 2007.

[28] H. Yin and D. Song. Temu: Binary code analysis via whole-system layered annotative execution. Technical Report UCB/EECS-2010-3, EECS Department, University of California, Berkeley, Jan. 2010. URL http://www.eecs.berkeley.edu/Pubs/TechRpts/2010/EECS-2010-3.html.

[29] M. Zhang, X. Zhang, X. Zhang, and S. Prabhakar. Tracing lineage beyond relational operators. In *Proceedings of VLDB'07*, Vienna, Austria, Sep. 2007. URL http://dl.acm.org/citation.cfm?id=1325851.1325977.

[30] J. Zhao, C. Goble, R. Stevens, and D. Turi. Mining Taverna's semantic web of provenance. *Concurr. Comput.: Pract. & Exper.*, 20(5):463–472, Apr. 2008. ISSN 1532-0634. . URL http://dx.doi.org/10.1002/cpe.1231.